Virtual Test Automation Generator (VTAG)

Steven Singer
NAWCAD, Lakehurst, NJ 08733
Code 4813, Generic Technologies
Naval Air Warfare Center, Aircraft Division, Lakehurst
Phone (732)323-7072; email - singer@lakehurst.navy.mil

Larry Venetsky
NAWCAD, Lakehurst, NJ 08733
Code 4813, Generic Technologies
Naval Air Warfare Center, Aircraft Division, Lakehurst
Phone (732)323-7073; email - venetsky@lakehurst.navy.mil

Michael L. Lynch
Naval Undersea Warfare Center, Division Newport
Code 8313, Software Development
Newport, RI 02841-1708
(401) 841-7498 x 38986; email - lynchML@code831.npt.nuwc.navy.mil

ABSTRACT

For many years electronic design and test communities were disconnected entities. This paper presents a process that links design with test during the design phase. The process utilizes an automatic control system strategy to automate the design of test patterns for diagnostics. A basic control system includes a controller, plant and sensors. The target for this controller is the weighted sum of percent detection and the number of isolated faults. The components of the control system include a timed test proposer, Genetic Algorithm (test generator) and an unsupervised neural network called Fuzzy Adaptive Resonance Theory (FuzzyART) used as a "Virtual Sensor". The system being evaluated is simulated using design software (i.e., Verilog HDL or VHDL - digital and SPICE - analog). The paper will describe the architecture of the control system and will address details of the selection algorithms for the Genetic Algorithm. The paper will also describe the FuzzyART, why it was used and some key advantages. Finally, the paper includes some results obtained for three different circuits (two digital and one analog) evaluated with this process.

Background

This paper describes a system that was envisioned to help couple the design and test efforts during the development of electronic systems. Test and diagnostic capability is typically developed after the design is complete, often by a different organization. In most cases, diagnosis of the Unit Under Test (UUT) is extremely difficult because the optimal test sites are not available at the edge connector. Developing the test and diagnostics during the design cycle could significantly improve diagnostic accuracy (reduce false pulls), reduce test time and considerably reduce costs both in development and maintenance. The intent is to increase the availability of the system in the field while reducing life-cycle costs.

In the past, the method of generating a test program with its associated diagnostics relied on the use of a fault simulator by a test engineer. The engineer used the fault simulator to model the circuit and manually add test patterns to a sequence called the input pattern set (IPS). During the design of the IPS, the engineer used a fault simulator recursively try new patterns appended to the previous patterns. The fault simulator assessed the fault coverage based on detection and a statistical evaluation of occurrence to interpret isolation. Ultimately, the fault simulator provides, when the test engineer is complete, a portion of the Test Program Set (TPS), which includes the IPS, the diagnostic information (fault dictionary), and some of the control software for the Automatic Test Equipment (ATE). This method currently requires a significant amount of a test engineer's time for a UUT of moderate complexity.

Obviously, the TPS is not presently developed during the design phase of the system due to time manpower, and cost constraints. The software required for fault simulation is not the same as that required for design and it is costly. If the designer builds the fault dictionary and test sequence during the design phase, improvements to fault detection and isolation can be made by literally

designing the test points that will be provided as output pins at the edge connector. Presently, the designer makes an "educated guess" as to which internal states require access at the edge connector based on a priori knowledge of the circuits and systems.

For design engineers to become involved with test engineering, the process that generates test sequences must be automated, reliable and predictable. The automated tools which synthesize the tests must use available standard design simulation languages (e.g., Verilog HDL and VHDL for digital circuits). The last crucial requirement is that the time to build a good test must be reasonable and fit within the current design cycle.

System Identification-Intelligent Control

The desire is to treat the circuit as a black box (see Figure 1) considering only input and outputs obtained at the edge connectors. This is how tests are implemented by ATE, and this is also how the interpretation of the results is accomplished. A test sequence must be generated and supplied to the UUT. Once this is accomplished, the output must be evaluated to identify the actual system. System Identification techniques apply this same philosophy.

The basic objective of system identification is to form a mathematical equivalent to an unknown physical system. There are four requirements for System Identification to occur. They are: define the general physical properties of the internal structures, sufficiently excite those internal structures, measure the response with proper sensors, and provide adjustments to the equivalent system (model) to match performance. One can imagine a set of springs coupled together as in Figure 1. The object of this experiment would be to find an input or set of inputs that would cause each spring to vibrate at its natural frequency which would allow for reconstruction of the spring constants by exciting each spring at their own resonant frequency. Ultimately, the equivalent system would develop over time so that each spring would be clearly identified.

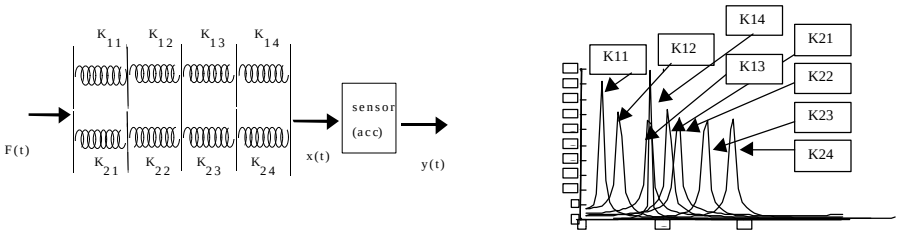


Figure. 1: Example of system identification.

The example above simplifies the reality of the true physics of this procedure, but it provides a clear understanding of the intent. Consider the distance moved by each spring to be a displacement defined as an internal state. The idea is to measure the state or motion at the output and mathematically reconstruct the internal states of the springs. In electrical circuits, the states typically are voltage, current (continuous or discrete).

It is not a trivial task to apply directly the concept of system identification to develop diagnostics. Typically, the matrices that develop using the standard approach become intolerably large and prohibit the use of this methodology. However, altering the internal states one at a time from its nominal value and representing this situation as a separate model essentially serves the same

purpose. Figure 2 introduces a system identification procedure that is designed to identify the faults, which are analogous to the states of the springs.
The block diagram of Figure 2 depicts the process of this Automatic Test Generator (ATG) system. This ATG is composed of four basic functional elements: Controlled Test Proposer,

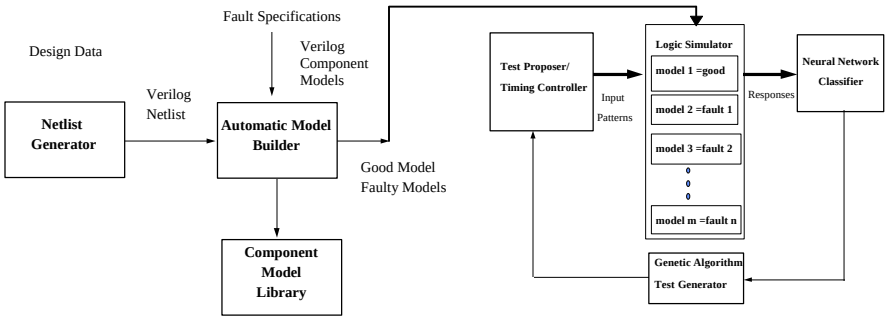


Figure 2. Automatic Test Generator and Automatic Model Builder

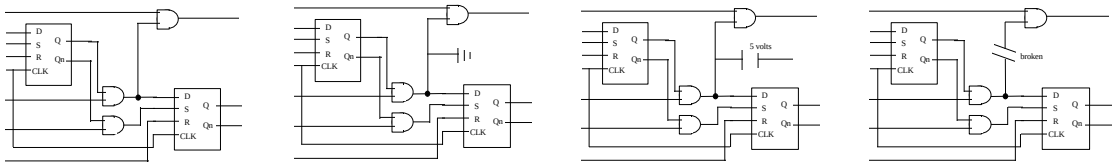
Simulator, Unsupervised Neural Network Classifier (NNC) and the Genetic Algorithm Test Generator. The Automatic Model Builder (AMB) consists of three basic structures, Netlist

Generator (NG), Component Model Library (CML), and fault specifications. The AMB converts computer-aided design (CAD) or schematic capture data into a fully functional circuit model.
The AMB utilizes the models in the Component Model Library (CML) and a netlist to assemble the software component models as they are wired together in hardware. Once the functional circuit model is validated, single faults are systematically inserted into the model to simulate failure conditions. The ATG system shown in Figure 2 is intended to minimize the number of input patterns (test vectors) that are required in order to achieve maximum fault detection and isolation.

Fault Models

A complete description of the fault models for both digital and analog circuits can be found in the literature (Lynch, 1997). A brief overview is provided in this section.

Fault Models (Digital). Presently, the type of faults that are used as models are of three types which are depicted in Figure 3. The first is each network of wires tied together by a common voltage source is fixed at a value equivalent to ground or 0. The second model is all networks wired together by a common voltage source is fixed at a value equivalent to rail voltage (dependent on technology) considered stuck at 1. The third type of fault model is a fan in terminal is separated from the voltage reference. This is considered a value of 1 as well. Each of these fault types is considered one at a time. This is to essentially highlight the voltage path so the test can be evaluated against detection and isolation of that fault (observability).



a) good system b) stuck-at-0 c) stuck-at-1 d) a broken pin

Figure 3. Representative models

A complete description of fault model construction for digital circuits in the Verilog HDL is described in the literature (Lynch and Singer, 1997).

Fault Models (Analog). For analog circuits, the faults that are modeled are: shorted pins, opened pins, resistors opened, capacitors opened and shorted, diodes shorted or opened, and the op-amp outputs stuck at positive rail or negative rail These faults are consistent with the faults to be detected on the Consolidated Automated Support System (CASS) as specified in the General Acceptance Test Procedures (GATP) for Operational Test Program Sets (OTPS), which is part of the CASS Red Team Generic Procurement Package. It is anticipated that all of the component failure modes defined in the GATP could be accommodated. In general, any fault which can be modeled can be processed by the system.

Simulator

A Verilog simulator, such as SILOS® III, (or SPICE, for analog circuits) exercises each of the circuit models (good and faulty) once for each individual test in the population during each iteration of the Genetic Algorithm (GA). A simulator-specific batch file must be created which invokes each circuit model simulation in turn. Each circuit uses the same IPS proposed by the GA and outputs responses to a different output file. After the execution of each individual test in the population, the output responses are stored until the complete set of individual tests in the population has been processed. The response data is then used by the Neural Network Classifier to perform its function. The system has developed tests using Simucad's SILOS® III, commonly-used, commercial off the shelf Verilog simulator.

NOTATION. The notation for the mathematics used by the system is as follows:

Inputs

$$\text{Population} = \begin{vmatrix} \text{Test}_1 \\ \text{Test}_2 \\ \text{Test}_3 \\ \cdot \\ \text{Test}_s \end{vmatrix}$$

where
s = number of tests sequences in the population;
x = 1, 2, ..., s;
j = 1, 2, 3, . . . , number of test patterns;
and
p = number of inputs to the circuit.
$\therefore$(Population is an s by n*p matrix)

$\text{Test}_x = [P_{x1}, P_{x2} \ldots, P_{xj}]$

$P_{xj} = [I_1, I_2, I_3, \ldots I_p]$

Prior to each test sequence an initialization sequence is inserted in the circuit simulations to initialize the system.

Outputs

Output matrix =

$$\begin{vmatrix} O_{11}, O_{12}, O_{13}, ..., O_{1y} \\ O_{21}, O_{22}, O_{23}, ..., O_{2y} \\ O_{31}, O_{32}, O_{33}, ..., O_{3y} \\ . \\ . \\ . \\ O_{m1}, O_{m2}, O_{m3}, ..., O_{my} \end{vmatrix}$$

where
m = 1, 2, 3, . . ., number of models,
with m=1 representing the good model response, m=2 representing the first fault model response, etc.
and y = input pattern number.

O_{my} is the response of the model m to input pattern y.

$Output_m = [O_{m1}, O_{m2}, O_{m3}, ..., O_{my}]$

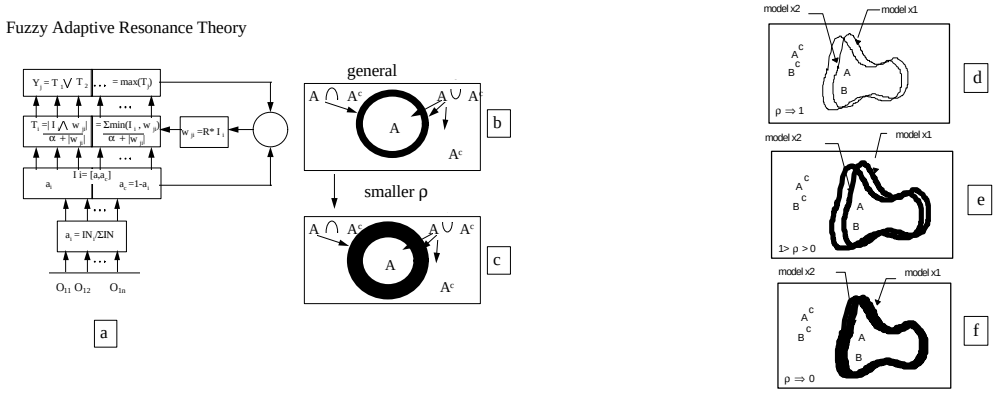
One additional conversion implemented in this data structuring for the digital circuit was to convert the output from a binary string to a set of integers using the following: O =[...0 1 0] = [..0*2^2+1*2^1+0*2^0]. This operation was done to reduce the amount of data, increasing the speed of analysis.

Unsupervised Pattern Classifier

The main feature of the classifier shown in Figure 2 is to generate the fitness function. The algorithm implemented in this process was developed by Steven Grossberg and Gail Carpenter at Boston University. The algorithm essentially uses the output matrix as weights with scalar multiples that define the sensitivity to pattern variations. As each row is multiplied by the rows of the weight matrix, a vector is formed, labeled T that contains results for this multiplication. The entry with the maximum value in this vector indicates the matched patterns.

The process isolates different patterns and groups them to form clusters or classes. For each test, a maximum number of clusters are formed, which indicates the number of fault classes isolated. The numbers of rows that are listed as class 1 (the good circuit class) are used to determine the percentage of fault detection. Finally, the weighted sum of both percent detection and total classes formed (number of faults isolated) becomes proportional to the fitness of each individual test.

Description Of FuzzyART. Fuzzy ART is a real number classifier. Because of this, it will be able to process digital, analog, and mixed signal circuits. FuzzyART uses fuzzy logic calculations to determine the relationship of memorized images to the image presented for recognition. An overview is provided in Figure 4. The weighted vectors stored in memory represent the patterns to be recognized. The intersection of patterns form a region of uncertainty that presents difficulty for the standard classifiers. This region of uncertainty, for FuzzyART can actually work to its advantage. Each pattern to be classified is provided as an input. The neural network then processes the signal and compares it to stored weights. If there is a match to one of the rows of the weighting matrix, the process stops. However, if there is no match the weight matrix is updated by $w_{ji}=\rho*I$ (I is the pattern to be classified modified with its complement) and used for future classifications. This process is repeated until the output matrix (used as input to the neural network) is exhausted. The scalar ρ is used to increase or decrease the resolution of the classifier making it ideal for analog and digital type systems. A complete description of both fuzzy logic and FuzzyART can be found in (Kosko, 1991) and (Grossberg, 1992), respectively.



computational flow **Boundary between a set and its complement increases as of ρ decreases** **progressive loss individuality as ρ ⇒ 0.**

Figure 4. Overview of FuzzyART

Two important points must be made on behalf of the benefits of FuzzyART for this process. Since FuzzyART is inherently unsupervised, it can identify patterns that are different thus separating the response of fault models into separate classes. The classifier which is used to generate tests off-line can be also used during testing to update the list of isolated faults that are diagnosed in the field. This is considered to be a real time machine learning process.

The second benefit is in reference to a particular challenge that analog circuits present. The testing of analog circuits has a region of uncertainty because of tolerances of components and the instruments evaluating the response. Since FuzzyART takes advantage of fuzzy set theory, it adds the benefits of improving the analysis of signals that are contained in the uncertainty boundaries. One way to take advantage of this network is to reduce the parameter ρ (increasing the size of the membership function formed by the patterns to be classified (Grossberg, 1992) so that only circuits within the tolerance limits are defined as acceptable. Once the parameter ρ is determined, the process is activated with the reduced value of ρ and a test is formed to isolate features that are characteristic of a faulty system. This concept can only be implemented with a mathematical structure that evaluates the region bounded within intersections of classes. Figure 5 illustrates the concept of using the region of uncertainty to extract features that are slightly different from the good circuit response making it possible to identify parametric faults.

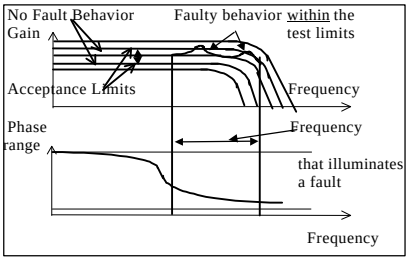


Figure 5. Uncertainty Regions

Test Proposition And Genetic Algorithm Test Generator

In any optimization process, potential solutions to the problem must be created and the performance for each attempted solution must be obtained and compared. A new set of potential solutions must be assembled and evaluated. This process must be repeated until the solution reaches the desired goals.

The Genetic Algorithm (GA) works in a manner described above. The GA is based on Charles Darwin's Evolutionary Theory of Natural Selection. This algorithm was first formally modeled mathematically by John Holland at the University of Michigan (Goldberg, 1989).

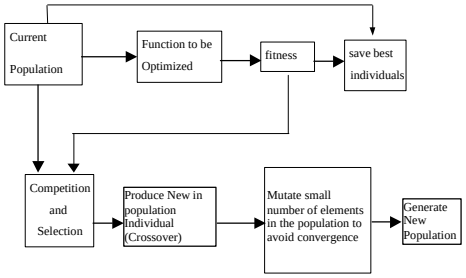


Figure 6. Block diagram of one loop of the GA

Applied to this ATG process, the GA process shown in Figure 6 generates new tests based on the old tests and their measured performance. The steps are; generate a population of tests either randomly or by a priori methods (Backtrace algorithm or verification test), select tests from the population, and evaluate each individual to determine its worth for selection. Selection is based on a statistical criteria which is modeled as either a "tournament" or a voting strategy. Once two tests have been selected, a crossover operation can be applied to create a new set of patterns.

Crossover. The method of generating a new test is to split each selected individual into two or more segments. Two new tests are formed by recombining (crossover) the front and back of each test where the front segment from test 1 becomes the front of test 2. The basic concept of the genetic algorithm is that, during crossover each test contributes a portion that had benefit to the fitness. By recombining components of the original test, new tests are formed that improve performance over several generations. Figure 7 illustrates an example of crossover. The actual system implemented a two point crossover.

Old Test1 = [**a b c d e f** g h i j k l] Old Test2= [m n o p q r **s t u v w x**]
New Test1= [**a b c d e f s t u v w x]** New Test2= [m n o p q r g h i j k l]

Figure 7. Example of crossover to form new tests.

Mutation. In addition to test generation by crossover, mutations to individual bits in each new test are performed by changing a small number of input values (typically less than 5%) from 0 to 1 or the reverse in a probabilistic fashion. This is necessary to sufficiently excite internal structures and maintain diversity.

Selection. The statistics defining the selection process are based on the probability of survival. Survival is determined by what has been termed "fitness function." The fitness function is the

workhorse that the GA requires for optimization. Fitness is the feedback that the process being optimized provides, via some knowledge of relative closeness to the desired goal. It is this feature that makes the GA distinctively different from other optimization schemes. The value of fitness does not necessarily require dependence on the models mathematically. It only requires performance of the system relative to the goal.

The mathematics behind the GA essentially determines, via the tournament selection process, those tests that will be selected for reproduction (survive) and those tests that will not be selected (die). This process is probabilistically based on the fitness function and ultimately pushes the average fitness of the groups of tests toward the detection/isolation goals from generation to generation by eliminating those tests that perform poorly.

The work described in this paper implemented two different methods of selection. The first was based on a tournament. Two individual tests were randomly selected. The values of their fitness were compared and the individual that won was selected for crossover based on a scalar defined by the user (known as probability of survival). The probability of survival (user defined parameter between 0 and 1) and is typically set to 0.75. A random number was generated between 0 and 1. If the random number was less than the probability of survival, the individual with the higher fitness was selected. The individual with the lower fitness was selected if the random number was greater than probability of survival. In other words, 75% of the time the individual with the better performance was selected and 25% of the time the other individual was selected.

The second selection methodology was based on the performance of each test. It is called Expected Value Model developed by Kenneth De Jong. A random number was generated between 0 and 1. Each individual was represented based on their fitness and compared with the mean of the entire population. Consider three individuals in the population: the first individual (I_1) has a fitness of 3, the second individual (I_2) has a fitness of 6, and the third individual (I_3) has a fitness of 9. The first individual with fitness 3 had a 16.7% chance of being selected. The individual with fitness of 6 had a 33.3 % chance of being selected. The last individual had a 50 % chance of being selected. In general, the probability that a test is selected is given by $P_i=f_i/(\sum f_i)$; where f_i = fitness for each individual test in the current population. A complete evaluation of the construction and mathematics of the genetic algorithm can be found in (Goldberg, 1989).

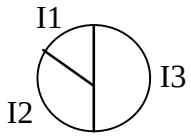


Figure 8: Selection based on Expected Value Model

Selective Breeding. In addition to the selection algorithms described above, a separate selection algorithm, designed to enhance the performance of the Genetic Algorithm, was developed. A new population was formed using the best performer from a previous number of generations. This improved overall performance and also helped reduce the required size of the population. Figure 8 illustrates an example of this selective breeding selection.

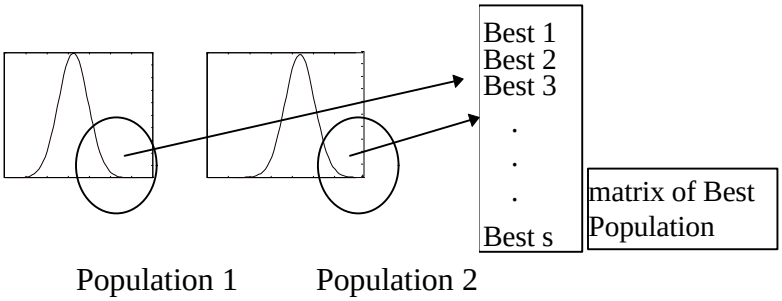
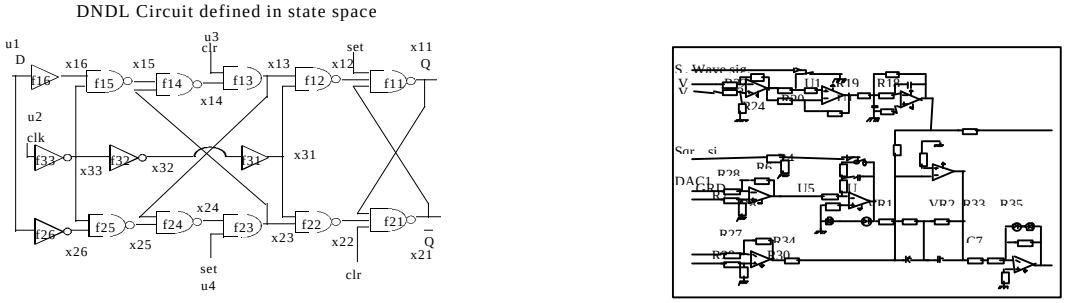


Figure 9. Example of selective breeding

Results

In order to assess the performance of this approach, the process was applied to three separate circuits. The first circuit was the D-type Non-inverting Data Latch (DNDL) of Figure 9. The second circuit was a moderately complex decoder (used in military aircraft) including flip-flops, counters, shift registers among other time dependent devices. The third circuit was the servo controller for a rate gyro (used in military aircraft) shown in Figure 9. All three circuits have been modeled using different simulation engines. The first was a "home grown" simulator created in the modeling software language, "Matlab," and the second was either a commercial simulator developed by Simucad called, Silos III, for digital circuits or SPICE for analog circuits.

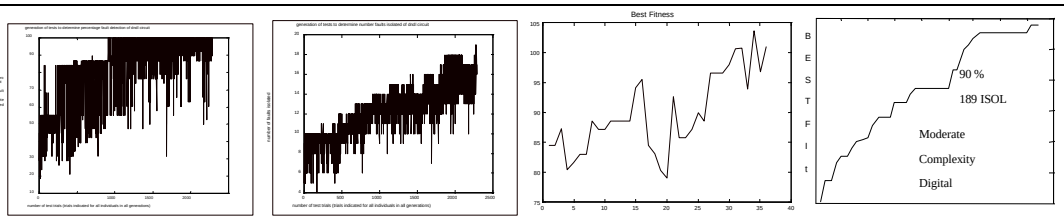
The results obtained for all three circuits were compared with existing analytical tools for fault analysis. The DNDL results indicated that the system achieved 100% detection and maximum available isolation (19 fault class equivalents) with 19 test patterns, compared to the back trace algorithm which achieved 100% detection and unknown isolation with 29 test patterns. The analog experiment achieved 100% detection with few generations. Isolation has only reached 60 % of the faults. This situation is still under investigation. Lastly, the decoder circuit which has a TPS developed manually reached the numbers achieved with the TPS with less than 100 test patterns as compared with 342 patterns in the TPS. At the time this paper was written, several experiments have been exercised to identify an optimized system.



(a) DNDL (b) rate gyro servo controller

Figure 10. Modeled Circuits

In addition, a hardware model of the DNDL circuit was built using discrete gate level components and faults were inserted at each pin. The Neural Network Classifier that was trained by the process outlined in this paper was built in an acquisition/control software system known as Labview and used for the diagnostics classifier. A similar Labview experiment is planned for the other two circuit models for early April 1998.



a) b) c)

Figure 11) Figure a represents the the DNDL percent detection and isolation respectively over the entire search cycle, b) represents the search improvements for the servo controller, and c) represents the Decoder circuit.

REFERENCES

Baker, J., 1989, An Analysis of the Effects of Selection In Genetic Algorithms, PhD Dissertation, University of Vanderbilt, Nashville, Tennessee.

Carpenter, G. and S. Grossberg, N. Markuzon, J. Reynolds and D. Rosen. "Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps." *IEEE Transactions on Neural Networks*, Vol3. No. 5, September 1992.

Goldberg, D., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc.

Kosko, B., 1991, *Neural Networks and Fuzzy Systems: A Dynamic Systems Approach to Machine Intelligence*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Lynch, M.,1997, "Using Commercial Modeling And Simulation Tools Within NGTG", *Proceedings of the IEEE Autotestcon*, pp. 121-125.
7997, "A Next Generation ATPG System Using the Verilog HDL", *International Verilog HDL Conference Proceedings*, pp. 56 - 63.

Singer, S., 1996, "Implementation of Model-Based Intelligent Next Generation Test Generator Using Neural Networks." *Proceedings for The International Society for Optical Engineering (SPIE), Applications and Science of Artificial Neural Networks II*, Vol 2760, pp. 657-668.

Singer, S. and L. Venetsky, 1997, "Next Generation Test Generator (NGTG) for Analog Circuits", *Proceedings of the IEEE Autotestcon*, pp. 113-119.